Advanced Linear Regression

Gerardo De la O

2022‘

Question 11.1

This section explores several techniques that build upon basic regression models in order to improve the overall fit and explanatory power of the model.

Preparation

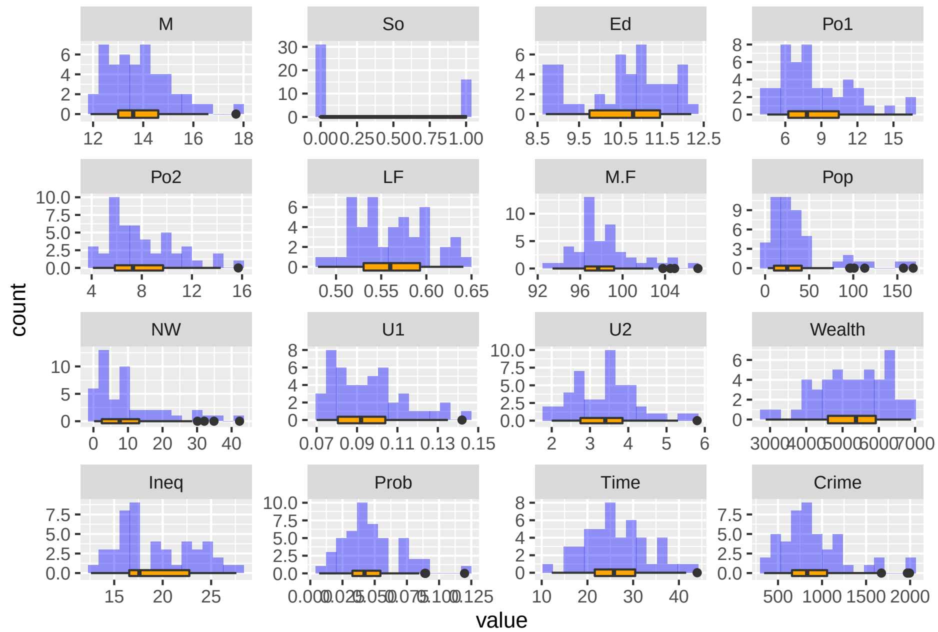
The purpose of this step is to understand the data and perform some basic cleanup. The graph below shows that we have 1 binary and 14 continuous predictors. Based on the distributions shown, we transform the data in the following manner:

1. Remove outliers (Values outside the 99.7th percentile)
2. Standardize the data to $\mu = 0$ and sd $= 1$

```
library(reshape2)
library(caret)

#import data
filename = 'http://www.statsci.org/data/general/uscrime.txt'
data <- read.csv(filename, sep = '\t' ) #tab delimited

# Melt to plot easily
melted_data <- melt(data)
ggplot(data=melted_data, aes(x=value)) + geom_histogram(bins=15, fill='blue',alpha=0.4) +
    geom_boxplot(fill='orange') + facet_wrap(~variable, scales = "free")
```




```
# Outlier removal
z_scores <- as.data.frame(sapply(data, function(data) (abs(data-mean(data))/sd(data))))
delete = rowSums(z_scores > 3)
df <- data[!delete, ]

# Scale the data
stdize = preProcess(df[,-16], method = c("center", "scale"))
df_sc = round(predict(stdize, df),4)

# Scale  and store the given input vector
testVal = df_sc[FALSE,-16]
testVal[nrow(testVal)+1,] = c(14,0,10,12,15.5,.640,94,150,1.1,.120,3.6,3200,20.1,.04,39)
testVal_sc = predict(stdize, testVal)
```

1. Stepwise Regression

Step-wise regression generates a model by adding and removing predictors until there is no valid reason to add or remove any more. Here, we apply stepwise regression starting with a model with no features.

In general, stepwise methods in R require a model with 0 features and a model with all features to be used as endpoints. The function step() does all the work for us afterwards! Important results from this analysis are summarized here:

1. RMSE on test set = 279.15
2. Features kept = 8
3. Prediction on test point = 1151

```
library(Metrics)
set.seed(1)

#Generate Train/Test Sets
smp_size <- floor(.80 * nrow(df_sc))
sampler <- sample(seq_len(nrow(df_sc)), size = smp_size)
train <- df_sc[sampler, ]
```

```r
test <- df_sc[-sampler, ]

# define "endpoint" models
zero <- lm(Crime ~ 1, data=train)
all <- lm(Crime ~ ., data=train)

# perform step-wise regression
stepReg <- step(zero, direction='both', scope=formula(all), trace=0)

# Fit model to test set
test_pred = predict(stepReg, test)

# Compute RMSE on test set
RMSE = rmse(actual = test$Crime, predicted = test_pred)
print(paste('RMS Error:', RMSE))
```

```
## [1] "RMS Error: 279.152482574631"
```

```r
# Model Details
print(summary(stepReg))
```

```
##
## Call:
## lm(formula = Crime ~ Po1 + Ineq + M + Ed + Prob + U2 + U1 + NW,
##     data = train)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -229.21  -89.40  -29.59   83.92  373.99
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)   916.20      27.86  32.886  < 2e-16 ***
## Po1           267.07      49.78   5.365 1.65e-05 ***
## Ineq          205.41      54.24   3.787 0.000901 ***
## M             135.18      39.71   3.404 0.002333 **
## Ed            237.59      52.35   4.538 0.000134 ***
## Prob         -106.29      40.02  -2.656 0.013833 *
## U2            159.54      59.08   2.700 0.012499 *
## U1            -91.91      52.30  -1.757 0.091595 .
## NW             61.49      44.25   1.390 0.177375
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 158.8 on 24 degrees of freedom
## Multiple R-squared:  0.8805, Adjusted R-squared:  0.8406
## F-statistic:  22.1 on 8 and 24 DF,  p-value: 2.729e-09
```

```r
# Test Input Prediction
TI = predict(stepReg, newdata=testVal_sc)
print(paste('Given Input Prediction: ', TI))
```

```
## [1] "Given Input Prediction:  1150.99803391815"
```

2. Lasso Regression

R's glmnet package applies the elastic net penalty (optimization) for a variety of glm objects. In this section we use glmnet to fit a model with the lasso constraint. For reference, the error function for gaussian elastic net is as follows:

$\frac{1}{2N}\sum_{i=1}^{N}(y_i - \beta_0 - x_i^T\beta)^2 + \lambda\left[(1-\alpha)\|\beta\|_2^2/2 + \alpha\|\beta\|_1\right],$

where $\lambda \geq 0$ is a complexity parameter that we have to tune ($\lambda = 0$ would be unpenalized regression) and $0 \leq \alpha \leq 1$ is a compromise between ridge regression (0) and lasso regression (1).

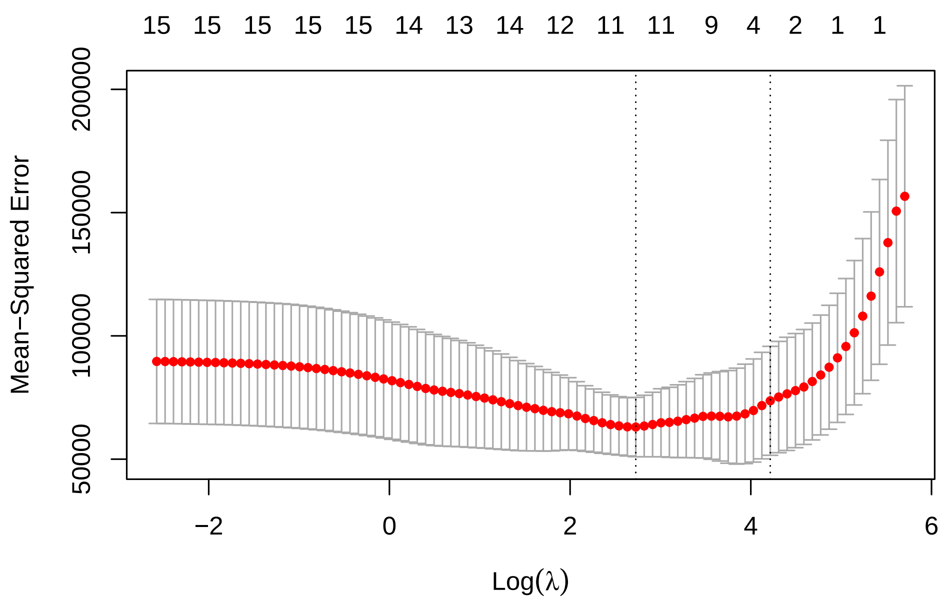
Important results from this analysis are summarized here:

1. Optimal Lambda = 15.3
2. RMSE on test set = 261 (better than step-wise regression)
3. Features kept = 11
4. Prediction on test point = 1373.2

```
library(glmnet)
set.seed(1)

train_x = as.matrix(df_sc[sampler,-16])
train_y = as.matrix(df_sc[sampler,16])
test_x = as.matrix(df_sc[-sampler,-16])
test_y = as.matrix(df_sc[-sampler,16])

# perform Lasso regression
cv_lasso <- cv.glmnet(train_x, train_y, family='gaussian', alpha=1,
                      type.measure='mse', nfolds=10, nlambda=100)
plot(cv_lasso, label=TRUE)
```




```
optimumLambda = cv_lasso$lambda.min
print(paste('Best Lambda:', round(optimumLambda,2)))
```

```
## [1] "Best Lambda: 15.29"
```

```r
# Fit model to test set
test_pred = predict(cv_lasso, newx = test_x, s = "lambda.min")

# Compute RMSE on test set
RMSE = rmse(actual = test_y, predicted = test_pred)
print(paste('RMS Error:', round(RMSE,2)))
```

```
## [1] "RMS Error: 261.04"
```

```r
# Model Details
coefs = as.matrix(coef(cv_lasso, s = 'lambda.min'))
print(coefs)
```

```
##                     s1
## (Intercept) 916.316406
## M            94.564436
## So           10.029632
## Ed          118.578460
## Po1         298.740325
## Po2           0.000000
## LF            0.000000
## M.F          10.888468
## Pop           5.689036
## NW           49.214992
## U1            0.000000
## U2           41.103156
## Wealth        0.000000
## Ineq        124.314778
## Prob        -46.231008
## Time         26.456757
```

```r
# Test Input Prediction
TI = predict(cv_lasso, newx=as.matrix(testVal_sc), s = "lambda.min")
print(paste('Given Input Prediction: ', round(TI,2)))
```

```
## [1] "Given Input Prediction:  1373.21"
```

3. Elastic Net Regression

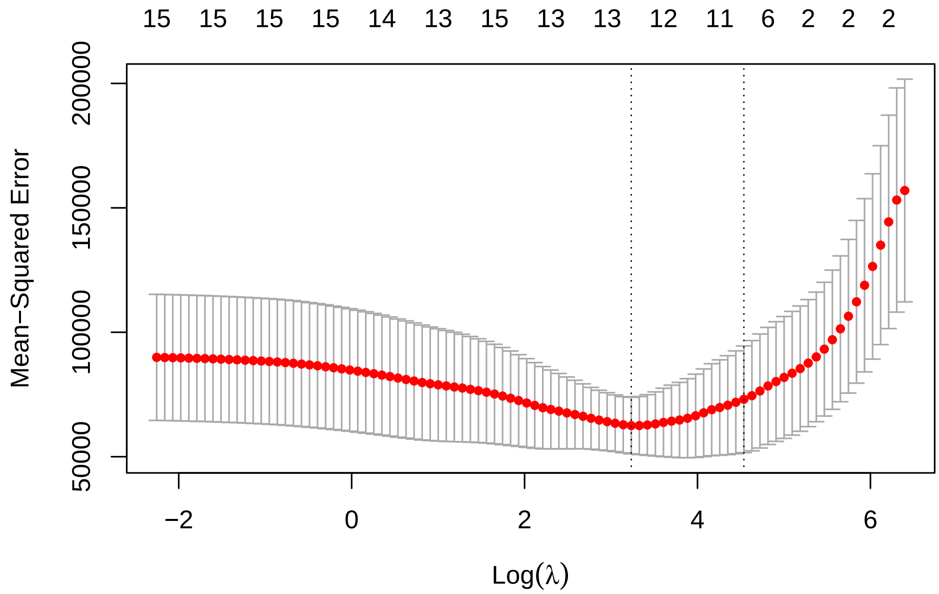
Continuing with the use of glmnet, we now focus on the Elastic-net method. In this section we use the package to fit a model with the elastic-net constraint. The method is essentially the same as Lasso regression, but we add some componenets if ridge regression. The end result is a model that should retain more coefficients than lasso.

Important results from this analysis are summarized here:

1. Optimal Lambda = 25.4
2. RMSE on test set = 266 (better than step-wise regression, worse than lasso)
3. Features kept = 12
4. Prediction on test point = 1535.6

```r
set.seed(1)

# perform Elastic Net regression
cv_lasso <- cv.glmnet(train_x, train_y, family='gaussian', alpha=.5,
                      type.measure='mse', nfolds=10, nlambda=100)
plot(cv_lasso, label=TRUE)
```




```
optimumLambda = cv_lasso$lambda.min
print(paste('Best Lambda:', round(optimumLambda,2)))
```

```
## [1] "Best Lambda: 25.38"
```

```
# Fit model to test set
test_pred = predict(cv_lasso, newx = test_x, s = "lambda.min")

# Compute RMSE on test set
RMSE = rmse(actual = test_y, predicted = test_pred)
print(paste('RMS Error:', round(RMSE,2)))
```

```
## [1] "RMS Error: 266.14"
```

```
# Model Details
coefs = as.matrix(coef(cv_lasso, s = 'lambda.min'))
print(coefs)
```

```
##                     s1
## (Intercept) 916.01289
## M            94.13869
## So           18.54685
## Ed          113.20838
## Po1         199.69738
## Po2          90.42990
## LF            0.00000
## M.F          25.41298
## Pop          13.32243
## NW           51.38880
## U1            0.00000
## U2           39.89801
## Wealth        0.00000
## Ineq        111.00194
```

```
## Prob        -45.05519
## Time         33.68920
```

```
# Test Input Prediction
TI = predict(cv_lasso, newx=as.matrix(testVal_sc), s = "lambda.min")
print(paste('Given Input Prediction: ', round(TI,2)))
```

```
## [1] "Given Input Prediction:  1535.59"
```

Final Remarks

Out of the 3 models built, Lasso regression had the best test set performance. However, step-wise regression was the easiest to generate and also has the most interpretable results. In practice, I would favor step-wise regression for exploratory analyses and one-off projects, whereas Lasso would be more favorable for a production model, where every bit of performance matters. It is worth noting that Elastic Net and Lasso are exactly the same method, with different values of alpha.